A Peer-to-Peer Communication Architecture for Networked Games

Shervin Shirmohammadi, Abdelfettah Diabi, Pascal Lacombe
Distributed Collaborative Virtual Environments Research Lab (DISCOVER Lab)
University of Ottawa, Ottawa, Canada
[shervin | adiabi]@discover.uottawa.ca

Abstract

Although IP multicast can be used to support message transmission among players in large network games running on Intranets, it is mostly not available on the Internet. Alternatively, researches have in recent years proposed the use of Application Layer Multicasting techniques (ALM) to alleviate this problem and to allow a somewhat scalable message passing among peers in a group of users on the Internet. In this paper, we propose a peer-to-peer communication architecture for networked games. Our architecture uses both proxies and end-systems to provide a number of communication services: 1) best effort LAN multicast 2) timely-reliable LAN multicast 3) best effort P2P delivery on the Internet 4) timely-reliable P2P delivery on the Internet, and 5) any combination of the above, including LAN to Internet translation and vice versa. Our filed trials using a shoot'em up game show that the proposed framework performs satisfactorily over the Internet.

1. Introduction

Like any other real-time network application, multiplayer networked games have specific requirements from their underlying transmission mechanism. In such games, players are required to participate in various activities in real time and to perform tasks in a synchronous manner, sometimes in a closely-coupled form that requires precise coordination between the parties, who otherwise are connected to the Internet from geographically distributed locations. It has been suggested that in such environments, the end-to-end delay should not exceed 100 msec [6]. Other studies have loosened this requirement to 200 msec as acceptable delay [4]. The reason for such strict requirement is the fact that such games, or distributed simulations, are naturally real-time and highly reactive multi-user processes where users interact based on the each other's actions and reactions; therefore requiring very low transmission delay of updates.

One of the problems, which has been studied and addressed to some extent in recent years, is network lag. It is a known fact that lag adversely affects networked games. When a user participates in such a game, his/her interactions with others users and/or the environment must be sent to other participants over the network, such that all entities involved are updated with that user's latest state. Because of network limitations and traffic conditions, some of these "updates" are lost, or delayed. In fact, network lag is present in any distributed application, such as web browsing, email, and audio/video streaming. However, due to its requirements for highly interactive operations, networked games are specially susceptible to packet loss and delay.

Much research has been done to compensate for network lag in order to provide better quality for distributed simulations. Some of these studies provide receiver-initiated and selectively-reliable transport protocols [5] that can be used to deliver important messages with a high degree of reliability, while others use sender-initiated approaches to transmit *key updates* with guaranteed reliability [8]. The IEEE DIS standard [3] has also been successfully used in controlled environment with vast resources, mostly for military simulations. These approaches; however, are all based on IP multicasting and although they achieve good results in Intranet environment, they are not readily deployable on the Internet.

The lack of applicability of IP multicasting on the Internet has been well documented [1][2]. Reasons include scalability, the fact that IP Multicasting is designed for a hierarchical routing infrastructure and does not scale well in terms of supporting large number of concurrent groups, the deployment hurdles caused by manual configuration at routers and Internet Service Providers' unwillingness to implement IP multicasting, and marketing reasons due to the undefined billing at the source (content provider) and receivers.

An alternative has therefore been proposed to shift multicast support from the networking layer to end systems. This is Application Layer Multicasting (ALM). In ALM, data packets are replicated at end-hosts instead of at routers. The end-hosts form an overlay network, and the goal of ALM is to construct and maintain an efficient overlay for data transmission. This is demonstrated in Figure 1. Since the routing information is maintained by the application, it is more scalable than IP multicasting since it can support large number of concurrent groups. Also, because ALM needs no infrastructure support, it is fully deployable on the Internet. In theory, Content Providers can deliver bandwidth-intensive contents such as TV programs and interactive networked games to vast number of clients via the Internet by using ALM. This was impractical before because the bottleneck bandwidth between content providers and consumers is considerably less than the natural consumption rate of such media.

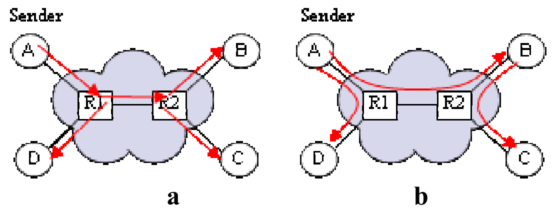


Figure 1. Network layer multicasting (a) versus application layer multicasting (b): square nodes are routers and circular nodes are end-hosts.

However, ALM does come with a tradeoff: more bandwidth and delay (compared to IP multicasting) for the sake of supporting more users and scalability. But it has been shown that ALM-based algorithms can have “acceptable” performance penalties with respect to IP Multicasting, and compared to other practical solutions [9].

In this article, we design, deploy, and test an ALM based protocol that can support networked games between players on the Intranet and on the Internet simultaneously. Our protocol, called the Hybrid Distributed Simulation Protocol (HDSP), supports both best-effort delivery, for frequently occurring messages, and reliable delivery for important or “key” messages. We use a sender-initiated approach to ensure guaranteed delivery of the key messages. Furthermore, we couple an in-house developed multiplayer game with our proposed protocol, and demonstrate that home users and LAN users can be supported simultaneously with satisfactory results. Due to usage of an appropriate ALM algorithm, our approach is more scalable than client-server approaches, while it is also more practical and deployable than IP-multicast based approaches.

2. Multi-user Networked Games

While a lot of studies have addressed many transport issues related to distributed simulations in general [10][11][12][13] [14], they fail to fulfill collaboration needs mainly because they do not consider the properties of collaboration data itself. The general assumption in distributed simulations is that objects transmit update messages often, and that the latest state of things can be determined by techniques such as dead-reckoning algorithms because they are somewhat predictable. Experience has shown that these assumptions work very well for scenarios such as simulation of battlefields or multi-user avatar-based games. In fact most of these systems have been specifically designed for either military purposes or games and they do a good job at that. In shoot-em-up games or battlefield scenarios, people, tanks, planes, and other war machines are almost constantly moving in a short-term predictable manner. A plane's course of flight can be extrapolated from its position and velocity vector. Also, a lost update message is usually followed by many other update messages, or keep-alive messages. However, these assumptions fail for Collaborative Virtual Environments (CVE). In fact in CVEs the conditions can be quite the opposite: shared objects often do not send continuous update messages, and when they do it is not necessarily in a predictable manner. When coordinating closely-coupled collaborative tasks in CVEs, there is no room for "guessing" the state of a shared object. All participants must reliably receive the most current state, or collaboration might fail. We believe a framework on top of which simulation applications are written must provide transport services suitable for all data: those that do not require reliability (regular updates) and those that do require reliability (key updates). With this in mind, let us have a closer look at how the Framework functions.

3. Communication framework

In this paper we propose HDSP (Hybrid Distributed Simulation Protocol) tailored to provide transport services suitable for all types of simulation data. HDSP is a multi-source protocol that provide best effort LAN multicast, timely-reliable LAN multicast, best effort P2P delivery on the Internet, timely-reliable P2P delivery on the Internet, and any combination of the above.

3. 1. Best effort LAN multicast

For data transmitted in the LAN, HDSP uses best effort LAN multicast mode if transmitted data does not require reliability. If data packets are not received by one or many users in the LAN then retransmission is not enforced. For this, HDSP simply implements normal UDP protocol.

3. 2. Timely-reliable LAN multicast

HDSP uses Timely reliable LAN multicast for data that must be received reliably and in a timely manner by all members. For this mode, an ACK-based reliable multicast protocol is utilized. When a packet is sent, a timer is invoked. Acknowledgements equal to the number of receivers should be received by the sender; otherwise the packet will be retransmitted after the timer expires.

3.3.Best effort P2P delivery on the internet

On the Internet, for data that must be received by a single known member in the group HDSP utilizes best effort P2P delivery. Data transmitted does not require reliability; therefore no retransmission is handled by this protocol. For this a normal UDP is used.

3.4. Timely-reliable P2P delivery on the Internet

For data that must be received by a single known member in the group in a timely manner, HDSP utilizes Timely-reliable P2P delivery. Data transmitted requires reliability; therefore retransmission is handled by this protocol in case of lost or damaged packets. For this mode, an ACK-based reliable protocol is utilized. When a packet is sent a timer is invoked. Acknowledgements equal to the number of connected peers should be received by the sender; otherwise the packet will be retransmitted after the timer expires.

3.5. Combination of the above services

The strength of HDSP appears clearly in incorporating ALM-tree to combine the above four services. HDSP is a multi-source protocol designed and implemented on top of a single multicast tree. The nodes of the tree consist of a Hybrid node which is placed in the LAN or on the Internet, other users in the LAN, and home users, as shown in figure 2. A hybrid node has all the information necessary to construct the ALM tree, maintain it and rearrange it in case of late joiners or early leavers. The new joiner send a request of joining to the Hybrid node (the root of the tree and the mediator between the tree and the LAN), who in return accepts it as a child node or redirects it to another node to be his parent. The process of accepting or redirecting a child node is based on the out-degree parameter. The out-degree parameter represents the maximum number of children a single node can have.

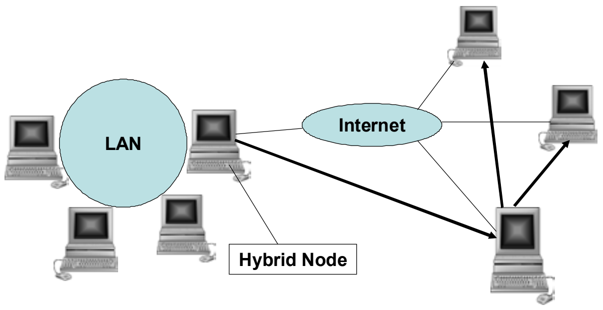


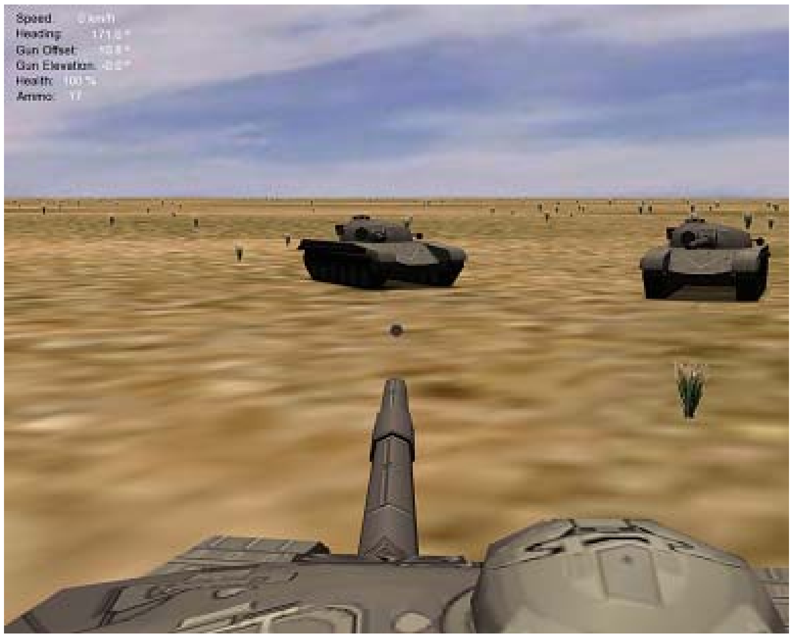
Figure 2. ALM tree construction. The thin lines indicate physical connection (LAN or Internet); whereas the thick lines indicate the overlay network, with the arrow going from parent to child.

After a tree is constructed, a message sent on the LAN is received by everyone on the LAN, including the Hybrid node through a multicast socket. The hybrid node will relay the message to its children, who in turn relay it to their children, and so on. HDSP provides support for both reliable and unreliable multicast. For unreliable multicast we use UDP multicast on LAN and unreliable ALM outside of LAN but for reliable multicast we use SCTP [9] on LAN and timely-reliable ALM (SCTP is implemented between children and parent) outside of LAN. The hybrid node will keep a record of the tree state and updates it every time when a user joins by checking every node in the tree for the possibility of having a child based on the out-degree. Every node in the tree will send a keep-alive message to its adjacent nodes to make sure that it did not crash or leave the session. In the case of no response to the keep-alive message then the node will have to relocate it self by sending a relocating request to the hybrid node.

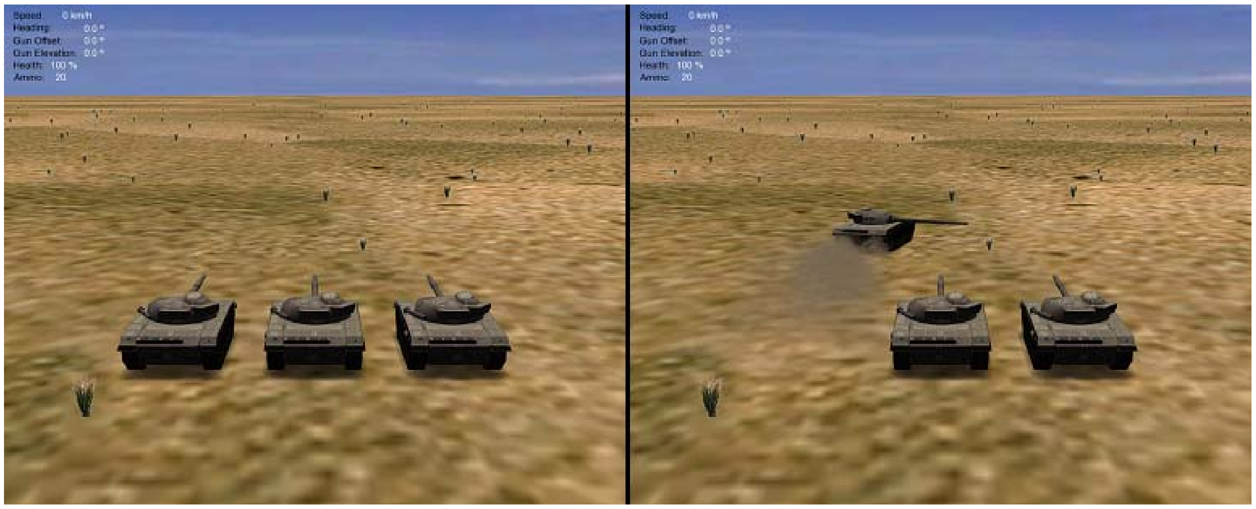
HDSP is basically an API-based communication framework that provides all of the above possibilities for communication. It is hybrid because we are using both a P2P architecture (home users between each other) and proxy to user architecture (Hybrid node to its children at home). Also, it is not mandatory to have users at home if it is only a LAN game, and it is not mandatory to have LAN players if this is only being played at home, in which case one of the players (whoever starts the game) can act as the hybrid node.

4. Game Design

Our main objective when we were designing our military tank simulation "Panzer Blaster" was to showcase the usability and advantages of HDSP in a distributed simulation environment. Since synchronization between users is critical for a decent experience in real-time distributed applications, HDSP would face an important test in terms of user end to end delay. We chose OpenSceneGraph (OSG), an object orientated OpenGL graphics library for C++, as our rendering toolkit since it's very well designed and uses a scene graph approach. This means the objects in our scene are placed in a tree for faster rendering and better data organization. Our scene contains four basic elements: a hud display, a terrain with vegetation, a sky and a list of users. Each user object contains a tank, a number of bullets and a list of variables needed to create a player.



a



b

Figure 3. a) A bullet is shot, this triggers a key message b) Movement messages are being sent, the tank moves accordingly and predicts the next movement

The “Panzer Blaster” simulation is run as a separate thread from the HDSP framework. It shares global buffers for receiving and sending messages. Those buffers are the only way in which the simulation needs to communicate with the HDSP framework. When a user wants to connect to a session, a key update message is sent to retrieve a user id from the server. This ID is then used to create and identify the user object for that specific player. The messages sent to move and connect a user are encoded bitwise into a string. Movement messages contain XYZ position, direction, speed and angle of the tank’s moving parts, while the shoot messages contain the XYZ origin and direction. All these messages also contain the type of message and the user id so each client knows which user to update. These messages are then put in the sending buffer and are specified as key or none-key messages depending on the situation. As depicted in figure3.a when a bullet is shot a key message will be triggered and immediately sent to adjacent users. Figure3.b shows a scenario of movement messages being sent to adjacent users. The receivers in this case will predict the next movement of the tank. The rest of the work for sending and redirecting is done by the HDSP framework. The receiving buffer is looked at every frame for new messages. They are decoded and applied to the scene as soon as they are received.

Every tank has a controller and a callback node to make it move. The callback nodes are called on a per frame basis during the traversal of the scene graph. The callback node then calls an update method in the controller to update the tank’s position if needed. The same principal is applied to move a bullet after its initial position has been calculated. When a message to move a tank is received, it’s only a matter of assigning its new position, speed and direction and the tank will follow this route until the next message is sent. To predict the tank’s movements, the previous message is stored. It’s assumed that the tank is either accelerating or stopping in a specific direction or axis if the new message has a different speed. If the message rate is higher then 3 per second, this prediction is rather accurate and is still reasonably good if only 1 message is sent per second

Messages are sent to adjacent nodes only when the tank is moving or shooting, eliminating the need for continuous bandwidth usage when nothing is happening. Key messages are only used for specific events, the rest are regarded as regular update messages and are simply broadcasted using UDP ALM. In our case key messages are used when one of the following events happen: stopping, shooting and the when moving from a stopped position. These messages are vital to reliably reproduce the movements of the tank on client machines. If we completely disabled these key messages, most of the time our tanks would continue moving for ever until the next message is sent. Sometimes the tank did stop approximately in the right spot but this was due to our movement prediction. As expected, some of the shooting messages where lost when not using key updates.

5. Performance Evaluation

Performance evaluation was conducted in three different homes having internet cable connection (two computers were using Rojers connection in Ottawa and one with videotron in Hull-Gatineau). One machine was acting as the hybrid node. We set the out-degree to be one such that we have two levels tree.

The Hybrid node sends messages every second for a period of 5 minutes, resulting in 300 packets sent. The receiving node forwards the packet to its children and acknowledges the messages by sending an acknowledgment to the parent. Each parent should forward the acknowledgment message to his parent until the hybrid node receives the acknowledgment from all users. The Hybrid node will measure the round trip time of the packet which is:

RTT = (time Ack is received - time packet was sent)/2

This is shown in figure 4.

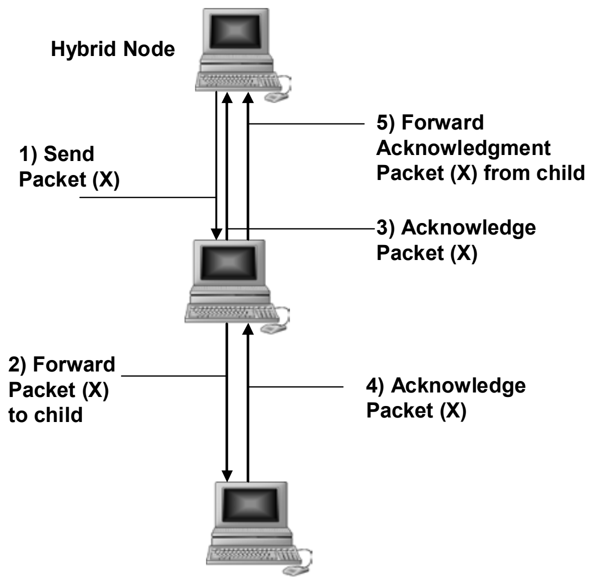

Hybrid Node
1) Send Packet (X)
5) Forward Acknowledgment Packet (X) from child
3) Acknowledge Packet (X)
2) Forward Packet (X) to child
4) Acknowledge Packet (X)

Figure 4. Test procedure for a period of 5mins

In table 1 we show the statistics that were measured during the test:

Table 1. Average delay values in milliseconds.

Number of pakets	Delay average to level 1(ms)	Delay average to level 2(ms)	Delay average for all nodes in the tree(ms)
300	38.30	130.6	84.45

5.1 Analysis

Analyzing the results we can see that the processing time in the first child before forwarding the packet and also the processing time of the ISPs have an impact of the transmission delay. We deduce that 2- levels are easily supported without violating the 200 ms end-to-end delay. It is also obvious that the processing time because of different ISPs has an impact on the delay from one node to another. There for we deduce the following:

1. In case the nodes are under the same ISPs, level 3 will be easily supported and the expected delay from the Hybrid node to the 3ed level is about 120-140 ms. It is not a coincidence that even level 4 will be supported with HDSP.

2. In case nodes are under different ISPs, HDSP we will definitely support the third level with the maximum average delay around the threshold (200msec) or less. However, level 4 might not be supported.

5.2 Subjective Evaluation

As expected the playing experience of “Panzer Blaster” on a LAN was very good. Since the delay was minimal all the messages got sent with constant and minimum latency. As we continued to add multiple levels (up to 6), we couldn’t perceive any difference in the end to end delay of each user. The playing experience didn’t change much when we tested the simulation between home users. We couldn’t say for sure whether the end to end delay affected our playing but it was very playable and similar to the LAN experience. We did notice more warping then on the LAN but this is attributable to the simulation itself and among other things its lack of smoothing when applying the messages.

Conclusions

In this article, we demonstrated how application layer multicasting can be used in conjunction with sender-initiated reliable communication to support large-scale networked games on the Internet. Our performance evaluation results showed that 3 levels of users can be supported in our architecture, with many users being able to connect at each level.
Other than the related IP-multicast based protocols mentioned, Mauve et al [6] present an architecture that uses proxies to provide fairness between players, low latency, congestion control, and robustness. However, our work differs in that HDSP categorize the messages as regular or key update messages which is an essential part in providing reliability for important messages and reducing network congestion by acknowledging only key messages. Our work also differs in a sense that the Hybrid node has the tree structure, regulates the growth of the tree by assigning new children to their prospective parents and rearranges the tree in case of nodes leaving the tree or crashing.

Acknowledgements

The authors acknowledge the financial support of Natural Sciences and Engineering Research Council of Canada.

References

[1] A. El-Sayed and V. Roca, “A Survey of Proposals for an Alternative Group Communication Service,” IEEE Network, 17(1), pp. 46-51, Feb. 2003.

[2] B. Zhang, S. Jamin, and L. Zhang, “Host multicast: A framework for delivering multicast to end users,” In Proc IEEE INFOCOM, June 2002.

[3] IEEE Standard for Distributed Interactive Simulation, Application Protocols, IEEE 1278-1995.

[4] K.S. Park and Robert V. Kenyon, "Effects of Network Characteristics on Human Performance in a Collaborative Virtual Environment", IEEE International Conference on Virtual Reality (VR '99), Houston, Texas, March 1999.

[5] M Pullen, "Reliable Multicast Network Transport for Distributed Virtual Simulation", Proc. IEEE Workshop on Distributed Interactive Simulations and Ral-Time Applications (DIS-RT '99), Greenbelt, Maryland, October 1999.

[6] M. Mauve, S. Fischer, and J Widmer, "A generic proxy system for networked computer games", Netgames 2002, Braunschweig, Germany, 2002.

[7] M.M. Wloka, "Lag in Multiprocessor VR", Presence: Teleoperators and Virtual Environments (MIT Press), Vol. 4, No. 1, Spring 1995.

[8] S. Shirmohammadi and N.D. Georganas, "An End-to-End Communication Architecture for Collaborative Virtual Environments", Computer Networks, Vol. 35, No. 2-3, Feb. 2001, pp. 351-367.

[9] Y. Chu, S.G. Rao, S. Seshan, and H.S. Zhang, "A Case for End System Multicast", IEEE Journal on Selected Areas in Communication, special issue on networking support for multicast, 2002, Volume 20, Issue 8, pp. 1456-1471.

[10] B. Blau et al, "Networked Virtual Environments", Proc. ACM SGGRAPH, 1992, pp. 157-164.

[11] T. Funkhouser, "Network Topologies for Scalable Multi-User Virtual Environments", Proc. IEEE Virtual Reality Annual International Symposium, 1996, pp. 222-228.

[12] M. Pullen et al, "Limitations of Internet Protocol Suite for Distributed Simulation in the Large Multicast Environment ", RFC 2502, February 1999.

[13] M.R. Macedonia, and M.J. Zyda, "A Taxonomy for Networked Virtual Environments", IEEE Multimedia Magazine, January-March 1997, pp. 48-56.

[14] S. Seidensticker et al, "Scenarios and Appropriate Protocols for Distributed Interactive Simulation", Internet draft, draft-myjak-lsma-scenarios-02, march 1997.